**COSC 264**

**Introduction to Computer Networks**

**and the Internet**

**Assignment**

Liz Richardson

Stefan Hall

Due: September 9

*"""*

*Sender part of assignment*

*#--Liz And Stefan--#*

*To run me type in the command line:*

*python sender.py portNum PortNum PortNum fileName*

*"""*

*# python3 sender.py 7001 7000 8000 test.txt*

**import** sys

**import** socket

**import** os.path *#This is being used to check if file exists*

**import** struct

**import** binascii

**import** packets

**import** select

MAX\_BYTES = 512

PTYPE\_DATA = 0

PTYPE\_ACK = 1

MAGICNO = hex(0x497E)

**def** exit():

*"""Exits the program displaying a message"""*

**print**(**"----------------------------------------"**)

**print**(**"------------Program exited--------------"**)

**print**(**"----------------------------------------"**)

sys.exit(0)

**def** checkPort(num):

*"""checks port number to ensure that it is an integer in the range 1024 - 64000*

*exits program if it finds and error*

*returns portnumber as an integer"""*

**try**:

port\_num = int(num)

**except** ValueError:

**print**(**"Port Number "** + str(num) + **" was not an integer"**)

exit()

**if not**(1024 <= port\_num **and** port\_num <= 64000):

**print**(**"Port number "** + str(num) + **" was not in range 1024 - 64000"**)

exit()

**else**:

**return** port\_num

**def** checkFile(fname):

*"""checks for existing file*

*exits if file not found"""*

**if**(os.path.isfile(fname)):

**return** fname

**else**:

**print**(**"File: "** + fname + **" does not exist."**)

exit()

**def** get\_params():

*"""gets all of the parameters from the command line*

*exits if wrong amount of arguments are entered*

*checks to ensure input is correct*

*returns port numbers and filename"""*

**if** len(sys.argv) != 5:

**print**(**"\nWrong amount of command line arguments entered"**)

exit()

**else**:

sys.argv = sys.argv[1:]

sin = checkPort(sys.argv[0])

sout = checkPort(sys.argv[1])

csin = checkPort(sys.argv[2])

filename = checkFile(sys.argv[3])

**return** sin, sout, csin, filename

**def** read\_file\_data(file\_obj):

*"""reads a max amount of data into a buffer"""*

**return** file\_obj.read(MAX\_BYTES)

**def** return\_resources(socket\_list, file\_object):

*"""closes sockets and file*

*giving memory back to system"""*

**for** sockets **in** socket\_list:

sockets.close()

file\_object.close()

**def** raise\_socket\_error(socket\_list, file\_object, ERROR\_COUNT):

*"""if connection to a socket is refused an error is raised*

*will wait 10 times and then time out"""*

**if** ERROR\_COUNT >5:

**print**()

**print**(**"----------------------------------------"**)

**print**(**"-----------Connection timeout-----------"**)

**print**(**"----------Program will now exit---------"**)

**print**(**"----------------------------------------"**)

**print**()

return\_resources(socket\_list, file\_object)

exit()

**else**:

**print**()

**print**(**"---connection refused, trying again-----"**)

**print**()

**return** ERROR\_COUNT+1

**def** main():

*"""main function for sender*

*sends a file to channel"""*

sin, sout, csin, filename = get\_params()

sockOut = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockOut.bind((**'127.0.0.1'**, sout))

sockOut.connect((**'127.0.0.1'**, csin))

sockIn = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockIn.bind((**'127.0.0.1'**, sin))

next = 0

exitFlag = False

file\_object = open(filename, **"rb"**)

all\_packets\_count = 0

ack\_packet\_count = 0

socket\_list = [sockOut, sockIn]

ERROR\_COUNT = 0

**print**()

**while not** exitFlag:

current\_string = read\_file\_data(file\_object)

**if** len(current\_string) == 0:

current\_packet = packets.packet(next, len(current\_string), current\_string)

exitFlag = True

**elif** len(current\_string) > 0:

current\_packet = packets.packet(next, len(current\_string), current\_string)

encoded\_packet = packets.pack\_packet(current\_packet)

has\_response = False

**while not** has\_response:

**try**:

sockOut.send(encoded\_packet) *#sendencoded packet*

all\_packets\_count += 1

**except**:

ERROR\_COUNT = raise\_socket\_error(socket\_list, file\_object, ERROR\_COUNT)

**else**:

ERROR\_COUNT = 0 *#a packet has been successfully sent, reset errorcount*

readable, \_, \_ = select.select([sockIn], [], [], 1)

**if** readable:

data = sockIn.recv(528)

unpacked\_packet = packets.unpack\_packet(data)

magicno, type, seqno, dataLen, byte\_data = unpacked\_packet

**if** magicno == int(MAGICNO, 0) **and** type == PTYPE\_ACK **and** dataLen == 0:

**if** seqno == next:

**if** exitFlag:

has\_response = True

next ^= 1

ack\_packet\_count +=1

has\_response = True

**print**(**"Total num packets sent: "**, all\_packets\_count)

return\_resources(socket\_list, file\_object)

exit()

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

main()

*"""*

*Channel part of assignment*

*#--Liz And Stefan--#*

*To run me type in the command line:*

*python channel.py portNum PortNum PortNum PortNum S(in) R(in) P*

*# python3 channel.py 8000 8001 8003 8002 7001 9000 0.1*

*"""*

**import** sys

**import** socket

**import** select

**import** packets

**import** random

**def** exit():

*"""Exits the program displaying a message"""*

**print**(**"Program will now exit\n"**)

sys.exit(0)

**def** checkPort(num):

*"""checks port number to ensure that it is an integer in the range 1024 - 64000*

*exits program if it finds and error*

*returns portnumber as an integer"""*

**try**:

port\_num = int(num)

**except** ValueError:

**print**(**"Port Number "** + str(num) + **" was not an integer"**)

exit()

**if not**(1024 <= port\_num **and** port\_num <= 64000):

**print**(**"Port number "** + str(num) + **" was not in range 1024 - 64000"**)

exit()

**else**:

**return** port\_num

**def** checkProbability(p):

*"""checks to make sure the probability enteed is a float in the range [0,1)*

*exits if there is an error*

*returns probability P """*

**try**:

p = float(p)

**except** ValueError:

**print**(**"Probability not an integer"**)

exit()

**else**:

**if not**(0 <= p < 1):

**print**(**"Probability not in range 0-1"**)

exit()

**else**:

**return** p

**def** get\_params():

*"""gets all of the parameters from the command line*

*exits if wrong amount of arguments are entered*

*checks to ensure input is correct*

*reutrns port numbers and P"""*

**if** len(sys.argv) != 8:

**print**(**"\nWrong amount of command line arguments entered"**)

exit()

**else**:

sys.argv = sys.argv[1:] *#chops off the name of the program*

csin = checkPort(sys.argv[0])

csout = checkPort(sys.argv[1])

crin = checkPort(sys.argv[2])

crout = checkPort(sys.argv[3])

sin = checkPort(sys.argv[4])

rin = checkPort(sys.argv[5])

P = checkProbability(sys.argv[6])

**return** csin, csout, crin, crout, sin, rin, P

**def** can\_send(P):

*"""generates a uniformly distrubuted number between 0 and 1*

*drops packet if u < P*

*returns true if packet can be sent"""*

to\_send = False

u = random.uniform(0, 1)

**if** u >= P:

to\_send = True

**return** to\_send

**def** return\_resources(socket\_list):

*"""closes sockets and file*

*giving memory back to system"""*

**for** sockets **in** socket\_list:

sockets.close()

**def** raise\_socket\_error(socket\_list, error\_count):

*"""If connection resets """*

**if** error\_count >5:

**print**()

**print**(**"----------------------------------------"**)

**print**(**"-----------Connection refused-----------"**)

**print**(**"----------Program will now exit---------"**)

**print**(**"----------------------------------------"**)

**print**()

return\_resources(socket\_list)

exit()

**else**:

**print**()

**print**(**"---connection refused, trying again-----"**)

**print**()

**return** error\_count + 1

**def** raise\_listen\_error(socket\_list, listening\_flag):

*"""Raises error if has been listening too long"""*

**if** listening\_flag > 5:

**print**()

**print**(**"----------------------------------------"**)

**print**(**"--------No responses in a while---------"**)

**print**(**"------ has been listening too long -----"**)

**print**(**"---------Program will now exit----------"**)

**print**(**"----------------------------------------"**)

**print**()

return\_resources(socket\_list)

exit()

**def** main():

*"""main fuction for channel*

*makes four sockets then listens waiting for a packet from sender or receiver*

*randomly decides to drop packet or not*

*if packet it isnt dropped it passes it through"""*

csin, csout, crin, crout, sin, rin, P = get\_params()

sockCSOut = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockCSOut.bind((**'127.0.0.1'**, csout))

sockCSOut.connect((**'127.0.0.1'**, sin)) *# So we don't have to specify where we send to*

sockCSIn = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockCSIn.bind((**'127.0.0.1'**, csin))

sockCROut = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockCROut.bind((**'127.0.0.1'**, crout))

sockCROut.connect((**'127.0.0.1'**, rin)) *# So we don't have to specify where we send to*

sockCRIn = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockCRIn.bind((**'127.0.0.1'**, crin))

socket\_list = [sockCROut, sockCRIn, sockCSIn, sockCSOut]

error\_count = 0

listening\_flag = 0

**while** True:

readable, \_, \_ = select.select([sockCSIn, sockCRIn], [], [], 1)

**print**(**"\nlistening\n"**)

**if** len(readable) == 0:

listening\_flag += 1

**if** listening\_flag > 20:

raise\_listen\_error(socket\_list, listening\_flag)

**else**:

listening\_flag = 0 *#reset listening time*

**for** sockets **in** readable:

**if** sockets **is** sockCSIn: *#if data is rome channel*

data, addr = sockCSIn.recvfrom(528)

unpacked\_packet = packets.unpack\_packet(data)

magicno, type, seqno, dataLen, byte\_data = unpacked\_packet

**if** packets.magicNoCheck(magicno):

**print**(**"recieved Datapack..."**)

**if** can\_send(P):

**print**(**"forwarding Datapack"**)

**try**:

sockCROut.send(data)

**except**:

error\_count = raise\_socket\_error(socket\_list, error\_count)

**else**:

**print**(**"PACKET FAILED TO SEND"**)

**if** sockets **is** sockCRIn: *#if data is from reciever*

data, addr = sockCRIn.recvfrom(528)

unpacked\_packet = packets.unpack\_packet(data)

magicno, type, seqno, dataLen, byte\_data = unpacked\_packet

**if** packets.magicNoCheck(magicno):

**print**(**"recieved ackPacket"**)

**if** can\_send(P):

*# now we can send the packet*

**print**(**"Forwarding AckPack"**)

**try**:

sockCSOut.send(data)

**print**(**"Ackpack sent"**)

**except**:

error\_count = raise\_socket\_error(socket\_list ,error\_count)

**else**:

**print**(**"ACKPACKET FAILED TO SEND"**)

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

*#makes it run automatically which is neat*

main()

*"""*

*Receiver part of assignment*

*#--Liz And Stefan--#*

*To run me type in the command line:*

*python3 receiver.py portNum PortNum PortNum fileName*

*# python3 receiver.py 9000 9001 8003 test.txt*

*"""*

**import** sys

**import** socket

**import** os.path

**import** packets

**import** select

MAX\_BYTES = 512

MAGICNO = hex(0x497E)

PTYPE\_DATA = 0

PTYPE\_ACK = 1

**def** exit():

*"""Exits the program displaying a message"""*

**print**(**"----------------------------------------"**)

**print**(**"------------Program exited--------------"**)

**print**(**"----------------------------------------"**)

sys.exit(0)

**def** checkPort(num):

*"""checks port number to ensure that it is an integer in the range 1024 - 64000*

*exits program if it finds and error*

*returns portnumber as an integer"""*

**try**:

port\_num = int(num)

**except** ValueError:

**print**(**"Port Number "** + str(num) + **" was not an integer"**)

exit()

**if not**(1024 <= port\_num **and** port\_num <= 64000):

**print**(**"Port number "** + str(num) + **" was not in range 1024 - 64000"**)

exit()

**else**:

**return** port\_num

**def** checkFile(fname):

*"""checks for existing file*

*exits if file not found"""*

**if**(os.path.isfile(fname)):

**print**(**"File: "** + fname + **" already exists."**)

exit()

**else**:

**return** fname

**def** get\_params():

*"""gets all of the parameters from the command line*

*exits if wrong amount of arguments are entered*

*checks to ensure input is correct*

*returns port numbers and filename"""*

**if** len(sys.argv) != 5:

**print**(**"\nWrong amount of command line arguments entered"**)

exit()

**else**:

sys.argv = sys.argv[1:]

rin = checkPort(sys.argv[0])

rout = checkPort(sys.argv[1])

crin = checkPort(sys.argv[2])

filename = checkFile(sys.argv[3])

**return** rin, rout, crin, filename

**def** return\_resources(socket\_list):

*"""closes sockets and file*

*giving memory back to system"""*

**for** sockets **in** socket\_list:

sockets.close()

**def** raise\_socket\_error(socket\_list, ERRORCOUNT=0):

*"""if connection to a socket is refused an error is raised*

*will wait 10 times and then time out"""*

**if** ERRORCOUNT>10:

**print**()

**print**(**"----------------------------------------"**)

**print**(**"-----------Connection refused-----------"**)

**print**(**"----------Program will now exit---------"**)

**print**(**"----------------------------------------"**)

**print**()

return\_resources(socket\_list)

exit()

**else**:

**print**()

**print**(**"---connection refused, trying again-----"**)

**print**()

**return** ERRORCOUNT+1

**def** main():

*"""Main function for reciever*

*recieves data and turns into file*

*sends ack packets"""*

rin, rout, CRIN, FILENAME = get\_params()

sockOut = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockOut.bind((**'127.0.0.1'**, rout))

sockOut.connect((**'127.0.0.1'**, CRIN))

sockIn = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sockIn.bind((**'127.0.0.1'**, rin))

expected = 0

recieving\_packets = True

socket\_list = [sockOut, sockIn]

error\_count = 0

file\_object = open(FILENAME, **"wb+"**)

**while** recieving\_packets:

readable, \_, \_ = select.select([sockIn], [], [], 1)

**if** readable:

**print**(**"recieved Pack"**)

data, sender = sockIn.recvfrom(528)

unpacked\_packet = packets.unpack\_packet(data)

magicno, type, seqno, dataLen, data = unpacked\_packet

**if** magicno == int(MAGICNO, 0) **and** type == PTYPE\_DATA **and** seqno == expected:

ack\_pack = packets.packet(seqno)

ack\_pack.set\_ack()

packed\_packet = packets.pack\_packet(ack\_pack)

expected ^= 1 *#Switches between 0 and 1 with XOR Using bitwise operator*

**if** dataLen > 0: *#data in packet*

file\_object.write(data)

**try**:

sockOut.send(packed\_packet)

**except**:

error\_count = raise\_socket\_error(socket\_list, error\_count)

**else**:

error\_count = 0 *#a packet has been successfully sent so we can reset errorcount*

**else**: *#datalen == 0*

ack\_pack = packets.packet(seqno)

ack\_pack.set\_ack()

packed\_packet = packets.pack\_packet(ack\_pack)

**try**:

sockOut.send(packed\_packet)

**except**:

error\_count = raise\_socket\_error(socket\_list, error\_count)

**else**:

error\_count = 0 *# a packet has been successfully sent so we can reset errorcount*

recieving\_packets = False

**elif**(magicno == int(MAGICNO, 0) **and** type == PTYPE\_DATA **and** seqno != expected):

ack\_pack = packets.packet(seqno)

ack\_pack.set\_ack()

packed\_packet = packets.pack\_packet(ack\_pack)

**try**:

sockOut.send(packed\_packet)

**except**:

error\_count = raise\_socket\_error(socket\_list, error\_count)

**else**:

error\_count = 0 *# a packet has been successfully sent so we can reset errorcount*

file\_object.close() *#returns resources*

**print**(**"RECIEVED ALL DATA"**)

exit()

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

*#makes it run automatically which is neat*

main()